GigaScience

ShinyLearner: A containerized benchmarking tool for machine-learning classification of tabular data

--Manuscript Draft--

Manuscript Number:	GIGA-D-19-00227R1
Full Title:	ShinyLearner: A containerized benchmarking tool for machine-learning classification of tabular data
Article Type:	Technical Note
Funding Information:	Brigham Young University Dr. Stephen R Piccolo
Abstract:	Classification algorithms assign observations to groups based on patterns in data. The machine-learning community have developed myriad classification algorithms, which are employed in diverse life-science research domains. When applying such algorithms, researchers face the challenge of deciding which algorithm(s) to apply in a given research domain. Algorithm choice can affect classification accuracy dramatically, so it is crucial that researchers optimize these choices based on empirical evidence rather than hearsay or anecdotal experience. In benchmark studies, multiple algorithms are applied to multiple datasets, and the researcher examines overall trends. In addition, the researcher may evaluate multiple hyperparameter combinations for each algorithm and use feature selection to reduce data dimensionality. Although software implementations of classification algorithms are widely available, robust benchmark comparisons are difficult to perform when researchers wish to compare algorithms that span multiple software packages. Programming interfaces, data formats, and evaluation procedures differ across software packages; and dependency conflicts may arise during installation. To address these challenges, we created ShinyLearner, an open-source project for integrating machine-learning packages into software containers. ShinyLearner provides a uniform interface for performing classification, irrespective of the library that implements each algorithm, thus facilitating benchmark comparisons. In addition, ShinyLearner enables researchers to optimize hyperparameters and select features via nested cross validation; it tracks all nested operations and generates output files that make these steps transparent. ShinyLearner includes a Web interface to help users more easily construct the commands necessary to perform benchmark comparisons. ShinyLearner is freely available at https://github.com/srp33/ShinyLearner.
Corresponding Author:	Stephen R Piccolo, Ph.D. Brigham Young University Provo, UT UNITED STATES
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Brigham Young University
Corresponding Author's Secondary Institution:	
First Author:	Terry J Lee
First Author Secondary Information:	
Order of Authors:	Terry J Lee
	Erica Suh
	Kimball Hill
	Stephen R Piccolo, Ph.D.
Order of Authors Secondary Information:	
Response to Reviewers:	Reviewer #1: The manuscript by Piccolo et al presents a new tool for comparing classification algorithms in a consistent interface. A key strength of the current approach is that it uses Docker containers so that once

a user has installed Docker, a wide variety of algorithms from different machine learning toolboxes can be compared. The typical user for this package would be someone new to machine learning wishing to compare a wide range of algorithms on their data.

>> Thank you for taking the time to write this review and for providing this positive feedback!

I am grateful to the authors for providing online resources; in particular the Code-Ocean capsule worked for me (after first logging in; presumably anonymous access is not possible), and all the figures from the manuscript were regenerated from the analysis results. (But if the online Rmd is to be updated, the output figures could be made taller as some of the y-axes were too squashed to see the range of data, e.g. I could not see the negative, red, differences for Figure 5 online).

>> Yes, as far as we understand it, anonymous access is not possible via Code Ocean. Thank you for the feedback about some of the figures being squashed. This is true for the HTML output file, produced from the Rmd file. However, we don't know of a way to control those dimensions. If you look at the right-hand panel of the output in Code Ocean, you should be able to see PDF files that are generated from the R code. The names of those files coincide with names specified in the Rmd file, so you should be able to match those and see what they look like in the correct dimensions.

What I couldn't do however, was to re-run the analysis. https://github.com/srp33/ShinyLearner/blob/master/Demo/Execute_Algorithms.ipynb lists the script, and it looks appropriate, but as I'm already late with this review, I'll have to assume it works. (How long does it take to execute?) Could it also be hosted on CODE OCEAN, or would it take too long?

>> It takes several days for this to execute. Although Code Ocean does not put hard limits on execution times, it's not really designed for such long-running executions.

The text from https://github.com/srp33/ShinyLearner/blob/master/Word_of_Caution.md is important and should be copied into the discussion section of the paper.

>> Thank you for this suggestion. We have added a slightly modified version of this text to the end of the Discussion section in our manuscript.

Although all the code is available on github, I think an archive of the github repo should be stored on Zenodo to give a permanent DOI of the repository when (assuming) the manuscript is published.

>> We have created Zenodo archives of the relevant GitHub repositories for this tool and have added the DOI for these archives to the manuscript under "Availability of source code and requirements".

The online tool http://bioapps.byu.edu/shinylearner/ looks great, but again due to being late with this review, I didn't get time to run it yet for myself. It would of course help for the very first time that there is a demo where I could download some data first (e.g. the Iris dataset, or MNIST) to work through this. Am I right in assuming that the role of the GUI is to build the eventual docker command to then be run locally?

>> You are correct that the role of the GUI is to build the eventual Docker command, which would be run locally. The GUI provides a link to a page on our GitHub site (https://github.com/srp33/ShinyLearner/blob/master/InputFormats.md), which describes input formats that can be used and provides example data files.

Minor:

I found it confusing to constantly flip between the main and supplementary figures. If a figure is important, please could it be folded into the main document? The first figure reference is on line 104, and that is to S1 (showing the docker command line invocation), rather than Figure 1 of the paper.

>> In the main part of the manuscript, we included what we thought would be figures that are most interesting to readers and relegated the remaining figures to the supplementary material. It just happened that the first figure we referenced was a supplementary figure, but the order in which the figures are mentioned does not necessarily indicate the figure's importance (as with any paper). Having said that, we can see the value in including the first supplementary figure in the main body, and we have made that change. When submitting our updated manuscript, we have uploaded the figures as separate files.

line 42: SUPPORT --> REQUIRE

>> This change has been made.

Is figure 2 required? It was obvious from figure 1 (to me at least) that the HoeffdingTree and decision_tree algorithms were lagging behind the others.

>> This figure has been moved to the Supplementary Material. We feel that it will be useful to some readers, even though it overlaps somewhat with the other figure.

lines 281-284: You show here that there are a few differences between algorithms that should be working the same. Did you explore why there were small differences? Parameter settings or initialisation methods? (I'm not surprised there are small differences, but thought you could explain them.)

>> We have added the following text for clarification: "For both of these algorithms, the available hyperparameters as well as options that users can select are considerably different between the underlying machine-learning libraries."

Figure 7: what classifier was used to do this analysis?

>> We have revised the caption for this figure to indicate that all 10 classification algorithms were used and that we averaged the results across these.

In the discussion, (line 389-396), six reasons supporting use of ShinyLearner are presented. I am convinced of the first two reasons, but I think most competent programmers would feel that they could also investigate points 3--6 in their own environment. Unless of course you are arguing that only ShinyLearner provides the wide diversity of algorithms that is absent in one environment (like R or Python). However, if you are to make this case, I think you need to point out specific examples of e.g. what classes of methods (rather than implementations) are missing e.g. in R or Python. My hunch, but happy to be proven wrong, is that R and Python each provide pretty much close to a full toolkit of machine learning methods.

>> To address these questions, we have added the following statement to the manuscript: "Although many of these tasks could be performed by a researcher who has programming expertise, care must be taken to ensure that the steps are performed in a robust manner (e.g., not mixing training and test sets in nested validation). In addition, we hope ShinyLearner will increase the efficiency of such benchmark studies by reducing duplicate efforts."

Figure S1: it looks like you are punching holes from Docker into the user's directory. I think you need to explain any potential security

risks here.

>> This should not be a security concern. To share data between the host operating system and the container, the user executing the container must have permission to read and write data in the host directories. Using commands such as those recommended by our GUI will ensure that the container can only read/write files for which the user has permission. There are other potential security concerns with Docker, but we feel that a discussion of those topics is outside the scope of this manuscript.

Figure S2: explain vertical dotted lines in legend.

>> We have added this to the caption.

Figure S3 (and S4): Are the Coefficients of Variation simply (s.d. / mean) or have they been multiplied by 100 to be a percentage?

>> The Coefficient of Variation is often expressed as a percentage in academic papers. We have modified the figure captions to clarify that these are percentages.

Figure S8: what does color denote?

>> We have clarified this in the caption.

Figure S10: Took me a while to work out the three coloured curves are for the three patients; perhaps rework last sentence of legend to make this clearer.

>> We have clarified this in the caption.

The word "Shiny" in the title should be explained somewhere to refer to the Shiny R package for making GUIs.

>> We have modified the text to explain this.

Stephen Eglen

Reviewer #2: The authors present a very compelling tool that allows researchers to compare and benchmark algorithms across various packages regardless of their design. Shinylearner also provides a simple web interface that allows users to easily generate Docker commands that anyone can execute (assuming they have a basic knowledge of Docker).

I found the rationale of the paper to be very clear and coherent. The authors make a clear case for the need for benchmarking, especially when comparing disparate algorithms and their various software implementations, especially from the lens of applying supervised learning to biomedical studies.

ShinyLearner is modular and extensive, and language agnostic, making it widely useful to a broad community of researchers. It also allows for diverse inputs and standardizes outputs. By running all benchmarks inside a container, it removes the challenges of dealing with complex dependencies, especially for users that may not be technically savvy, all the while not building out a black box. It supports a large number of classification algorithms, has gpu support and is already set up to be sustainable.

>> We thank the reviewer for these positive and encouraging comments!

Additional Information:	
Question	**Response**
Are you submitting this manuscript to a special series or article collection?	No

Experimental design and statistics Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our Minimum Standards Reporting Checklist. Information essential to interpreting the data presented should be made available in the figure legends. Have you included all the information requested in your manuscript?	Yes
Resources A description of all resources used, including antibodies, cell lines, animals and software tools, with enough information to allow them to be uniquely identified, should be included in the Methods section. Authors are strongly encouraged to cite Research Resource Identifiers (RRIDs) for antibodies, model organisms and tools, where possible. Have you included the information requested as detailed in our Minimum Standards Reporting Checklist?	Yes
Availability of data and materials All datasets and code on which the conclusions of the paper rely must be either included in your submission or deposited in publicly available repositories (where available and ethically appropriate), referencing such data using a unique identifier in the references and in the "Availability of Data and Materials" section of your manuscript. Have you have met the above requirement as detailed in our Minimum Standards Reporting Checklist?	Yes

ShinyLearner: A containerized benchmarking tool for machine-learning classification of tabular data

Stephen R. Piccolo[1,*], Terry J. Lee[1], Erica Suh[1], Kimball Hill[1]

1 - Department of Biology, Brigham Young University, Provo, UT, USA

* - Please address correspondence to S.R.P. at stephen_piccolo@byu.edu.

ORCID IDs:

Stephen R. Piccolo: 0000-0003-2001-5640; Kimball Hill: 0000-0001-9331-0777

Abstract

Classification algorithms assign observations to groups based on patterns in data. The machine-learning community have developed myriad classification algorithms, which are employed in diverse life-science research domains. When applying such algorithms, researchers face the challenge of deciding which algorithm(s) to apply in a given research domain. Algorithm choice can affect classification accuracy dramatically, so it is crucial that researchers optimize these choices based on empirical evidence rather than hearsay or anecdotal experience. In benchmark studies, multiple algorithms are applied to multiple datasets, and the researcher examines overall trends. In addition, the researcher may evaluate multiple hyperparameter combinations for each algorithm and use feature selection to reduce data dimensionality. Although software implementations of classification algorithms are widely available, robust benchmark comparisons are difficult to perform when researchers wish to compare algorithms that span multiple software packages. Programming interfaces, data formats, and evaluation procedures differ across software packages; and dependency conflicts may arise during installation. To address these challenges, we created ShinyLearner, an open-source project for integrating machine-learning packages into software containers. ShinyLearner provides a uniform interface for performing classification, irrespective of the library that implements each algorithm, thus facilitating benchmark comparisons. In addition, ShinyLearner enables researchers to optimize hyperparameters and select features via nested cross validation; it tracks all nested operations and generates output files that make these steps transparent. ShinyLearner includes a Web interface to help users more easily construct the commands necessary to perform benchmark comparisons. ShinyLearner is freely available at https://github.com/srp33/ShinyLearner.

Keywords: Machine learning, supervised learning, classification, software containers, benchmark, feature selection, algorithm optimization, model selection

Background

Classification falls under the category of supervised learning, a branch of machine learning. When performing classification, researchers seek to assign observations to distinct groups. For example, medical researchers use classification algorithms to identify patterns that predict whether patients have a particular disease, will respond positively to a particular treatment, or will survive a relatively long period of time after diagnosis[1–11]. Applications in molecular biology include annotating DNA sequencing elements, identifying gene structures, and predicting protein secondary structures[12].

Typically, a classification algorithm is “trained” on a dataset that contains samples (observations) from two or more groups, and the algorithm identifies patterns that differ among the groups. If these patterns are reliable indicators of group membership, the algorithm will be able to accurately assign new samples to these groups and thus may be suitable for broader application. Different research applications require different levels of accuracy before classification algorithms are suitable for broader application. However, even small improvements in accuracy can provide large benefits. For example, if an algorithm predicts drug-treatment responses for 1000 patients and attains accuracy levels that are 2% higher than a baseline method, this algorithm would benefit 20 additional patients. Accordingly, a key focus of classification research in the life sciences is to identify generalizable ways to optimize prediction accuracy.

The machine-learning community have developed hundreds of classification algorithms and have incorporated many of these implementations into open-source software packages[13–18]. Each algorithm has different properties, which affect its suitability for particular applications. In addition, most algorithms require hyperparameters, which alter the algorithms' behavior and can affect the algorithms' accuracy dramatically. In addition, feature-selection (or feature-ranking) algorithms can be used in complement to classification algorithms, helping to identify combinations of variables that are most predictive of group membership and aiding in data interpretation[19,20]. With this abundance of options to consider, researchers face the challenge of identifying which algorithm(s), hyperparameter combinations, and features are optimal for a particular dataset.

To improve the odds of making successful predictions, researchers should choose algorithms, hyperparameters, and features based on empirical evidence rather than hearsay or anecdotal experience. Prior studies can provide insight into algorithm performance, but few studies evaluate algorithms comprehensively, and performance may vary widely for different types of data. One way to select these options empirically is via nested cross-validation[21]. With this approach, a researcher divides a single dataset into training and validation sets. Within each training set, the researcher divides the data further into training and validation subsets and then evaluates various options using these subsets. The top-performing option(s) are then used when making predictions on the outer validation set. Alternatively, a researcher might perform a benchmark study, applying (non-nested) cross validation to multiple datasets from a given research domain. After testing multiple algorithms, hyperparameters, and/or feature subsets, the researcher can examine overall trends and identify options that tend to perform well[22,23]. With either approach, it is ideal to evaluate a comprehensive set of options. However, several challenges make it difficult to perform such evaluations effectively:

- Researchers may wish to compare algorithms that have been implemented in different software packages. Although many machine-learning packages allow users to execute algorithms programmatically, application programming interfaces (APIs) are not standardized, and they are implemented in diverse programming languages.
- Different software implementations use different techniques for evaluating algorithm performance, so it is difficult to ensure that comparisons are consistent.
- Input and output formats differ by software implementation, thus requiring custom efforts to prepare data and interpret results.
- When installing the software, researchers typically must install a series of software dependencies. Installation requirements often differ by operating system, and versioning conflicts can arise[24].

To reduce these barriers, we created ShinyLearner. For this open-source project, we have integrated existing machine-learning packages into containers, which provide a consistent interface for performing benchmark comparisons of classification algorithms. ShinyLearner can be installed on Linux, Mac, or Windows operating systems, with no need to install software dependencies other than the Docker containerization software. ShinyLearner currently supports 53 classification algorithms and 1300+ hyperparameter combinations across these algorithms; users can perform automatic hyperparameter tuning via nested cross validation. In addition, ShinyLearner supports 16 feature-selection algorithms, enabling researchers to reduce data dimensionality before performing classification (via nested cross validation). New algorithms can be integrated in an extensible manner.

ShinyLearner is designed to be friendly to non-computational scientists—no programming is required. We provide a Web-based tool (https://bioapps.byu.edu/shinylearner) to guide users through the process of creating the Docker commands necessary to execute the software. ShinyLearner supports a variety of input formats and produces output files in “tidy data” format[25], thus making it easy to import results into external tools. Even though other machine-learning packages support nested cross validation, these evaluations may occur in a “black box.” ShinyLearner tracks all nested operations and generates output files that make this process transparent.

Below we describe ShinyLearner in more detail and illustrate its use via benchmark evaluations. We evaluate 10 classification algorithms and 10 feature-selection algorithms on 10 biomedical datasets. In addition, we assess the effects of hyperparameter optimization on predictive performance, provide insights on model interpretability, and consider practical elements of performing benchmark comparisons.

Methods

ShinyLearner org (RRID: SCR_017608) encapsulates open-source, machine-learning packages into Docker images[26], which are available on Docker Hub (https://hub.docker.com/r/srp33/shinylearner/). Currently, ShinyLearner supports algorithms from scikit-learn, Weka, mlr, h2o, and Keras (with a TensorFlow backend)[13–15,27–29]. To facilitate user interaction, to harmonize execution across the tools, and to evaluate predictive performance, ShinyLearner uses shell scripts, Python scripts, R scripts, and Java code[30–32]; these are included in the Docker images. To perform an analysis, the user executes a shell command, specifying arguments to indicate the location(s) of the input files, which algorithms to use, whether to perform Monte Carlo or k-fold cross validation, etc. The analysis is executed within a container, and output files are saved to a directory that the user specifies. TensorFlow provides support for execution on graphical processing units, which requires a slightly different software configuration, so we provide a separate Docker image that enables this feature (https://hub.docker.com/r/srp33/shinylearner_gpu/). All changes to the ShinyLearner code are tested via continuous integration[33]; build status can be viewed at https://travis-ci.org/srp33/ShinyLearner.

Figure 1 shows an example ShinyLearner command that a user might execute. For convenience, and to help users who have limited experience with Docker or the command line, we created a Web-based user interface where users can specify local data paths, choose algorithms from a list, and select other settings (https://bioapps.byu.edu/shinylearner). After the user has made these selections, the Web interface generates a Docker command, which the user can copy and paste; Windows Command Line, Mac Terminal, and Linux Terminal commands are generated. We used the R Shiny (RRID:SCR_001626) framework to build this web application[34], hence the name *ShinyLearner*.

ShinyLearner interfaces with each third-party machine-learning package via shell scripts wrap that around the software's API. For each algorithm, one shell script specifies the algorithm's default hyperparameters. In most cases, additional shell scripts specify alternative hyperparameters. The classification algorithms in ShinyLearner span methodological categories, including linear models, kernel-based techniques, tree-based approaches, Bayesian models, distance-based methods, ensemble approaches, and neural networks. In selecting algorithms to include, we focused primarily on implementations that can handle discrete and continuous data values, support multiple classes, and produce probabilistic predictions. For each algorithm, we reviewed documentation for the third-party software and identified a representative variety of hyperparameter options. Admittedly, these selections are somewhat arbitrary and inexhaustive. However, they can be extended with additional options. We excluded some algorithm implementations and hyperparameter combinations because errors occurred when we attempted to execute them or because they failed to achieve reasonable levels of classification accuracy on simulated data.

Additional algorithms (and hyperparameter combinations) can be incorporated into ShinyLearner. The sole requirements are that they have been implemented as free and open-source software and provide an API (that can be executed via Linux command-line scripts). Users who wish to extend ShinyLearner must:

1. Identify any software dependencies that the new algorithm requires. If those dependencies are not currently included in the ShinyLearner image, the user must modify the ShinyLearner Dockerfiles accordingly.
2. Create bash script(s) that accepts specific arguments and invoke the new algorithm.

3. Request that these changes be included in ShinyLearner via a GitHub pull request.

ShinyLearner supports the following input-data formats: tab-separated value (.tsv), comma-separated value (.csv), and attribute-relation file format (.arff). When tab-separated or comma-separated files are used, column names and row names must be specified; by default, rows must represent samples (observations) and columns must represent features (variables). However, transposed versions of these formats can be used (features as rows and samples as columns); in these cases, the user should use ".ttsv" or ".tcsv" as the file extension. ShinyLearner accepts files that have been compressed with the gzip algorithm (using ".gz" as the file extension). Users may specify more than one data file as input, after which ShinyLearner will identify sample identifiers that overlap among the files and merge on those identifiers. If the user specifies, ShinyLearner will scale numeric values, one-hot encode categorical variables[35], and impute missing values.

ShinyLearner supports two schemes for evaluating predictive performance: Monte Carlo cross validation and k-fold cross validation[36,37]. In Monte Carlo cross validation, the data are split randomly into a training and validation set; the algorithm is allowed to access the class labels for the training data only. Later the algorithm makes predictions for the validation samples, and the accuracy of those predictions is evaluated using various metrics. Typically, this process is repeated many times to derive confidence intervals for the accuracy metrics. In k-fold cross validation, the process is similar, except that the data are partitioned into evenly sized groups and each group is used as a validation set through rounds of training and testing. When multiple algorithms or hyperparameter combinations are employed, ShinyLearner evaluates nested training and validation sets, with the goal of identifying the optimal combination for each algorithm. Then it uses these selections when making predictions on the outer validation set. Nested cross validation is also used for feature selection; a feature-selection algorithm ranks the features within each nested training set, and different quantities of top-ranked features are used to train the classification algorithm. The feature subsets that perform best are used in making the outer validation-set predictions.

Hyperparameter optimization and feature selection may be combined; however, such analyses are highly computationally intensive for large benchmarks.

All outputs are stored in tab-delimited files, thus enabling users to import results directly into external analysis tools. ShinyLearner produces output files that contain the following information for each combination of algorithm, hyperparameters, and cross-validation iteration: 1) predictions for each sample, 2) classification metrics, 3) execution times, and 4) standard output, including a log that indicates the arguments that were used, thus supporting reproducibility. When nested cross-validation is performed, ShinyLearner produces output for every hyperparameter combination that was tested in the nested folds and indicates which combination performed best for each algorithm.

ShinyLearner supports the following classification metrics:

- AUROC (Area under the receiver operating characteristic curve)[38]
- Accuracy (proportion of samples whose discrete prediction was correct)
- Balanced accuracy (to account for class imbalance)
- Brier score[39]
- F1 score[40]
- False discovery rate
- False negative rate
- False positive rate
- Matthews correlation coefficient[41]
- Mean misclassification error
- Negative predictive value
- Positive predictive value
- Recall (sensitivity)
- True negative rate (specificity)

- True positive rate (sensitivity)

To calculate these metrics and to perform other data-processing tasks, ShinyLearner uses the AUC[42], mlr[15], dplyr[43], data.table[44], and readr[45] packages. For multiclass problems, ShinyLearner allows the underlying machine-learning packages to use whatever strategy they have implemented for classifying with multiple classes. ShinyLearner then calculates performance metrics in a one-versus-rest manner and averages results across the class options.

When feature selection is performed, each algorithm produces a ranked list of features for each nested training set. To aid the user in understanding which features are most informative, ShinyLearner aggregates these ranked lists using the *Borda count* method[46]. These aggregate rankings are stored in tab-delimited output files.

The steps of preparing the data and executing ShinyLearner for the results described in this article are in a Jupyter notebook (see https://github.com/srp33/ShinyLearner/blob/master/Demo/Execute_Algorithms.ipynb). We used the ggplot2 and cowplot packages[47,48] to create figures.

Analyses

ShinyLearner enables researchers to perform classification benchmark studies. To illustrate this functionality, we performed three types of benchmark: 1) basic classification with default hyperparameters, 2) classification with hyperparameter optimization, and 3) classification with feature selection. For each analysis, we used 10 classification algorithms:

- `keras/dnn` - Deep neural networks (implemented in Keras/TensorFlow)[27,29,49]
- `mlr/h2o.randomForest` - Random forests (implemented in mlr, h2o)[15,28]
- `mlr/mlp` - Multilayer perceptron (mlr)[50]

- mlr/xgboost - xgboost (mlr)[51]
- sklearn/decision_tree - Decision tree (implemented in scikit-learn)[13,52]
- sklearn/logistic_regression - Logistic regression with the LIBLINEAR solver (scikit-learn)[53]
- sklearn/svm - Support vector machines (scikit-learn)[54]
- weka/HoeffdingTree - Hoeffding tree (implemented in Weka)[14,55]
- weka/MultilayerPerceptron - Multilayer perceptron (Weka)
- weka/SimpleLogistic - Simple logistic regression (Weka)[56]

In the third analysis, we used 10 feature-selection algorithms:

- mlr/kruskal.test - Kruskal-Wallis rank sum test (mlr)[57]
- mlr/randomForestSRC.rfsrc - Permuted random forests (mlr)[58]
- sklearn/mutual_info - Mutual information (scikit-learn)[59]
- sklearn/random_forest_rfe - Random forests—recursive feature elimination (scikit-learn)[60,61]
- sklearn/svm_rfe - Support vector machines—recursive feature elimination (scikit-learn)[61]
- weka/Correlation - Pearson’s correlation (Weka)[62]
- weka/GainRatio - Information gain ratio (Weka)[52]
- weka/OneR - OneR (Weka)[63]
- weka/ReliefF (Weka)[64]
- weka/SymmetricalUncertainty - Symmetrical uncertainty (Weka)[65]

In each analysis, we used 5 rounds of Monte Carlo cross validation. For the second and third analyses, we used 3 rounds of *nested* Monte Carlo cross validation for each *outer* round of cross validation. In the third analysis, we evaluated the top-ranked 1, 3, 5, 10, 15, 20, 50, and 200 features and identified the best of these options via nested cross validation. In evaluating the results, we focused on area under the receiver operating characteristic curve (AUROC) because this metric can be applied to probabilistic predictions and accounts for class imbalance.

As an initial test, we generated a “null” dataset using numpy[66]. We used this dataset to verify that ShinyLearner produces classification results in line with random-chance expectations when no signal is present. This dataset consisted of 20 numeric variables (mean = 0, standard deviation = 1) and 10 categorical variables across 500 simulated samples. AUROC values for all classification algorithms were near 0.5, as expected by random chance, irrespective of whether hyperparameter optimization or feature selection was performed (Figure S1).

Next, we collected 10 biomedical datasets from the Penn Machine Learning Benchmarks repository[67]:

- Acquired Immune Deficiency Syndrome (AIDS) categorical data[68]
- Thyroid disease[52]
- Breast cancer[69]
- Dermatology[70]
- Diabetes
- Hepatitis[71]
- Iris[72]
- Liver disorder[73]
- Molecular biology (promoter gene sequences)[74]
- Yeast[75]

These datasets vary by number of samples (minimum = 51; maximum = 7201) and number of features (min = 5; max = 172). For all datasets, we converted categorical variables to multiple binary variables using one-hot encoding. When executing ShinyLearner, we scaled numeric values using scikit-learn’s RobustScaler, which subtracts the median and scales the data based on the interquartile range[76];

accordingly, this method is robust to outliers. In addition, we used ShinyLearner to impute missing values; this method uses the median for numeric variables and the mode for categorical variables.

Classification analysis with default hyperparameters

Initially, we applied 10 classification algorithms to 10 biomedical datasets using default hyperparameters. Most algorithms made near-perfect predictions for the Thyroid, Dermatology, and Iris datasets, whereas predictions were less accurate overall for the remaining datasets (Figure 2). The `weka/HoeffdingTree` and `sklearn/decision_tree` algorithms often underperformed relative to the other algorithms (Figure S2). Indeed, for half of the datasets, `weka/HoeffdingTree` performed as poorly or worse than would be expected by random chance. The remaining 8 classification algorithms performed relatively well, but predictive performance varied considerably across the datasets (Figure S3). For example, the AUROC for `mlr/mlp` and `sklearn/logistic_regression` was 0.07 higher than the median on the AIDS dataset; the AUROC for `sklearn/svm` was 0.14 lower than the median.

Across the Monte Carlo iterations for each dataset, the predictive performance of `sklearn/decision_tree` and `weka/MultilayerPerceptron` varied most, whereas `weka/HoeffdingTree` varied least (in part because AUROC was frequently 0.5) (Figure S4). The `keras/dnn` and `mlr/h2o.randomForest` algorithms took longest to execute, whereas `sklearn/svm` and `sklearn/logistic_regression` were among the fastest (and most accurate) algorithms (Figure S5). Two pairs of classification algorithms use similar theoretical approaches but were implemented in different machine-learning libraries; multilayer perceptron was implemented in Weka and mlr; logistic regression was implemented in Weka and scikit-learn. The AUROC values were strongly—but not perfectly—correlated between these pairs of implementations (Figures S6 and S7). For both of these algorithms, the available hyperparameters as well as options that users can select are considerably different between the underlying machine-learning libraries.

With the exception of `sklearn/decision_tree`, all classification algorithms produced sample-wise, probabilistic predictions. We examined these predictions for the Diabetes dataset and found that the range and shape of these predictions differed widely across the algorithms (Figure 3). Although many classification metrics, including AUROC, can cope with distributional differences, these differences must be considered in multiple classifier systems[77].

Classification analysis with hyperparameter optimization

In the second analysis, we applied the same classification algorithms to the same datasets but allowed ShinyLearner to perform hyperparameter optimization via nested cross validation. As few as 2 (`mlr/xgboost`) and as many as 95 (`sklearn/decision_tree` and `weka/MultilayerPerceptron`) hyperparameter combinations were available for each algorithm. In nearly every example, classification performance improved after hyperparameter optimization (Figure 4), sometimes dramatically. The performance improvements were most drastic for the `weka/HoeffdingTree` and `sklearn/decision_tree` algorithms, which often performed poorly with default parameters.

ShinyLearner supports 53 hyperparameter combinations for the `keras/dnn` algorithm. Each of these combinations altered the algorithm's performance at least to a small degree on every dataset (Figure S8). The Thyroid dataset varied least across the hyperparameter combinations, perhaps because the number of instances (n = 7200) was nearly 10 times larger than any other dataset. Generally, this algorithm performed better with a wider architecture containing only two layers. Having a wider structure greatly increases the parameter space of the network and allows it to learn more complex relationships among features, while limiting the network to only two layers prevents overfitting, a common problem when applying neural networks to datasets with a limited number of instances. In addition, adding dropout and L2 regularization also helps to prevent the network from overfitting. In tuning these hyperparameters, we found that a smaller dropout rate, more training epochs, and a smaller regularization rate resulted in higher AUROC values (Figure S9). Figure S10 illustrates for the Diabetes dataset that diagnosis predictions can differ considerably, depending on which hyperparameter combination is used.

Classification analysis with feature selection

In any dataset, some features are likely to be more informative than other features. We used ShinyLearner to perform feature selection (via nested cross validation) before classification. In total, we evaluated 100 unique combinations of feature-selection algorithm and classification algorithm (with default hyperparameters). In 44% of cases, feature selection increased the median AUROC, whereas it decreased AUROC in 39% of cases (Figure 5). Feature selection sometimes improved the performance of `weka/HoeffdingTree` and `sklearn/decision_tree`, which were the lowest performers without feature selection.

Figure 6 illustrates the relative predictive ability of each combination of feature-selection and classification algorithms. The `mlr/randomForestSRC.rfsrc` and `sklearn/random_forest_rfe` algorithms performed best on average; both approaches use the Random Forests algorithm to evaluate feature relevance. The `weka/OneR` algorithm, which evaluates a single feature at a time in isolation, performed worst. Across the datasets, the combination of `mlr/randomForestSRC.rfsrc` (feature selection) and `mlr/xgboost` (classification) performed best. Perhaps surprisingly, the combination of `sklearn/svm_rfe` (feature selection) and `sklearn/svm` (classification), which are both based on Support Vector Machines, was ranked in the bottom quartile.

In seeking to identify the most informative features, ShinyLearner evaluated various quantities of top-ranked features via nested cross validation. Figure 7 illustrates the relative performance of each of these quantities on each dataset. In all cases but one, using one feature performed worst. Generally, a larger number of features resulted in higher AUROC values. However, more features sometimes decreased performance. For example, on the breast-cancer dataset, the highest AUROC values were attained using 3 out of 14 features.

ShinyLearner can inform users about which features are most informative for classification. In the Dermatology dataset, these feature ranks were highly consistent across the feature-selection algorithms (Figure S11). The goal of this classification problem was to predict a patient's type of Eryhemato-Squamous disease. Elongation and clubbing of the rete ridges as well as thinning of the suprapapillary epidermis were most highly informative of disease type, whereas features such as the patient's age were less informative.

Discussion

The machine-learning community has developed an abundance of algorithms and software implementations of those algorithms. Life scientists use these resources for many research applications. But they face the challenge of identifying which algorithms and hyperparameters will be most accurate and which features are most informative for a given dataset. Many researchers limit classification analyses to a single algorithm, perhaps one that is familiar to them or that has been reported in the literature for a similar study. Others may try a large number of algorithms; however, performing benchmark comparisons in an *ad hoc* manner requires a considerable coding effort and can introduce biases if done improperly. Alternatively, some researchers may develop new algorithms without providing evidence that these algorithms outperform existing ones. We developed ShinyLearner as a way to simplify the process of performing classification benchmark studies.

ShinyLearner does not implement any classification or feature-selection algorithm; rather, it serves as a wrapper around existing software implementations. Currently, algorithms from Weka, scikit-learn, mlr, h2o, and Keras are supported in ShinyLearner. In aggregate, these algorithms represent a diverse range of methodological approaches and thus can support comprehensive benchmark evaluations. On their own, each of the third-party tools encapsulated within ShinyLearner provides a way to optimize hyperparameters programmatically and perform feature selection. In addition, tools such as caret[17], KNIME[18], and Orange[78] provide these options. Thus, in situations where a researcher has programming expertise and is satisfied with the algorithms and tuning functionality available in one of those tools, the researcher might prefer to use these tools directly rather than use ShinyLearner. ShinyLearner is most useful when a researcher:

1. wishes to compare algorithms that have been implemented in multiple machine-learning packages,
2. does not have programming expertise,
3. desires to perform complex operations via nested cross validation, such as evaluating different sizes of feature subsets,
4. wishes to analyze algorithm performance using a tool or programming language that is different than was used to perform classification,
5. wishes to gain deeper insight into decisions made during nested cross validation, and/or
6. seeks to evaluate the tradeoff between predictive accuracy and time of execution.

Although many of these tasks could be performed by a researcher who has programming expertise, care must be taken to ensure that the steps are performed in a robust manner (e.g., not mixing training and test sets in nested validation). In addition, we hope ShinyLearner will increase the efficiency of such benchmark studies by reducing duplicate efforts.

ShinyLearner is limited to datasets that fit into computer memory. For larger datasets, frameworks such as Apache SystemML support distributed algorithm execution[79]; however, the number of algorithms implemented in these frameworks is still relatively small.

The current release of ShinyLearner supports diverse classification algorithms and hyperparameter combinations; however, this collection is far from exhaustive. Using ShinyLearner’s extensible architecture, the research community can integrate additional algorithms and hyperparameter combinations. In addition, algorithm designers can use our framework to compare their algorithms against competing methods and disseminate their algorithms to the research community.

Containers provide many advantages for software deployment. Tool installation and computational reproducibility are easier because all software components are encapsulated within the container, and container images can be archived and versioned[80]. One other benefit may be less apparent: containerization facilitates the use of diverse programming languages. Distinct components of ShinyLearner are implemented in 4 different programming languages. We chose this approach because we determined that each language was suited to specific types of tasks. We posit that the future of bioinformatics development will increasingly follow this pattern. Furthermore, we advocate for the approach of providing a graphical user interface, such as the Web-based tool we provided for ShinyLearner. Such tools make it easier for users—especially those who have limited command-line experience—to formulate Docker commands.

Our analysis of 10 biomedical datasets, 10 classification algorithms, and 10 feature-selection algorithms confirmed that the choice of algorithm and hyperparameters has a considerable impact on classification performance and selected features. Although some algorithms typically performed better than others, no single algorithm consistently outperformed any other. This finding supports the “No Free Lunch” theorem[81] and confirms that multiple classifier systems hold promise for aggregating evidence across algorithms[82]. Also importantly, algorithm performance is likely to differ according to data characteristics. Algorithms that perform well on “wide” datasets (many features, few samples) may not perform as well on “tall” datasets. Algorithms that perform well with numeric data may not perform as well on categorical or mixed data. These differences highlight the importance of domain-specific benchmark comparisons.

Finally, we offer recommendations regarding benchmark comparisons. When performing benchmarks across multiple algorithms and/or hyperparameters, it is important to exercise caution in interpreting those results. Below are recommendations on performing benchmarks and interpreting such results:

- If you apply multiple algorithms or hyperparameter combinations, you should always report those (e.g., in the Methods section of a journal article). It is poor form to report only the best results.
- After you have identified the best-performing algorithm and/or hyperparameters, it is usually best to test those findings on a completely independent dataset that was not used in the benchmark comparison.
- Merely because an algorithm (or parameter) appears to work well in one setting doesn't necessarily mean that the same will be true in alternate settings.

Availability of source code and requirements

- *Project name*: ShinyLearner
- *Project home page*: https://github.com/srp33/ShinyLearner
- *Operating system(s)*: Any operating system on which Docker can be installed
- *Programming languages*: Java, Python, R, bash
- *Other requirements*: Docker (https://docker.com)
- *License*: MIT
- DOI of *Zenodo* archives of GitHub repositories: 10.5281/zenodo.3543724, 10.5281/zenodo.3543726, 10.5281/zenodo.3543728, 10.5281/zenodo.3543730

The code for creating the figures in this manuscript can be found and re-executed in a Code Ocean capsule[83].

This tool has been registered at bio.tools (https://bio.tools/ShinyLearner) and at SciCrunch.org (RRID: SCR_017608).

Availability of Supporting Data

Archives of the GitHub repositories are available in Zenodo[84, 85, 86, 87]. Snapshots of all of the archives are also available in the *GigaScience* GigaDB repository[88].

Editors Note

A CODECHECK certificate for this paper is available confirming that the figures in the paper could be independently reproduced[89].

Figure Legends

Figure 1: Example ShinyLearner command for performing a benchmark comparison. In this example, the user wishes to place output files in a directory located at /home/user/OutputData. To avoid problems with file permissions, this directory should be created before Docker is executed. The docker run command builds a container and maps input and output directories from the host operating system to locations within the container (separated by colons). The --user directive indicates that the container should execute using the executing user's permissions. The name of the Docker image and tag name are specified (srp33/shinylearner:version511) as well as the name of a ShinyLearner script that performs nested, Monte Carlo cross validation (/UserScripts/nestedclassification_montecarlo). The remaining arguments indicate the name of the input data file, a description of the analysis, the number of Monte Carlo iterations, the classification algorithms, etc. ShinyLearner provides documentation on each of these arguments as well as a Web application for building such commands dynamically.

Figure 2: Classification performance per dataset (default hyperparameters). We evaluated the predictive performance of 10 classification algorithms on 10 biomedical datasets. These results were generated using default hyperparameters for each algorithm. We measured predictive performance using

the receiver operating characteristic curve (AUROC) and calculated the median across 5 Monte Carlo iterations. Predictive performance differed considerably across and within the datasets.

Figure 3: Sample-level predictions for each algorithm on the Diabetes dataset (default hyperparameters). The Diabetes dataset includes a class variable indicating whether or not patients received a positive diagnosis. Each panel of this figure shows positive-diagnosis predictions for each classification algorithm. All algorithms except `sklearn/decision_tree` produced probabilistic predictions. The range and distribution of these predictions differed greatly across the algorithms.

Figure 4: Classification performance when optimizing vs. not optimizing hyperparameters. We tested 10 classification algorithms on 10 biomedical datasets and used nested cross validation to select hyperparameters. To evaluate for change in predictive performance, we calculated the percent change in the median AUROC values when using optimized vs. default hyperparameters. Most algorithms demonstrated improved classification performance with optimized hyperparameters.

Figure 5: Classification performance when performing feature selection vs. not performing feature selection. In combination with classification, we performed feature selection via nested cross validation on 10 biomedical datasets. For each algorithm, we used default hyperparameters. These plots show the percent change in the median AUROC when using vs. not using feature selection. Although the effects of feature selection varied across the algorithms, median AUROCs increased in many cases.

Figure 6: Performance for each combination of classification and feature-selection algorithm. This figure shows classification results for the nested cross-validation folds across each combination of feature-selection algorithm and classification algorithm. Averaged across all datasets and classification algorithms, we ranked the feature-selection algorithms based on AUROC values attained for nested validation sets. For simplicity and consistency across the datasets, this figure shows only the results when the top-5 features were used. Higher average ranks indicate better classification performance.

Figure 7: Median classification performance of feature-selection algorithms by number of features. We applied feature selection to each dataset, in combination with each of the 10 classification algorithms. For each algorithm, we selected the top x number of features and averaged across each combination of feature-selection and classification algorithm. This figure shows which values of x resulted in the highest AUROC values for each dataset. Different datasets had different quantities of features; this graph only shows results for x values relevant to each dataset. Accordingly, we scaled the AUROC values in each column between zero and one to ensure that the comparisons were consistent across all datasets. Higher values indicate better classification performance. Generally, a larger number of features resulted in better classification performance, but this varied across the datasets.

Declarations

List of abbreviations

AUROC = Area under receiver operating characteristic curve

API = application programming interface

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Funding

SRP was supported by internal funds from Brigham Young University. TJL and KH were supported by fellowships from the Simmons Center for Cancer Research at Brigham Young University.

Author's contributions

SRP, TJL, and KH helped to develop the software. SRP conceived of the software design with critical input from TJL and KH. ES and SRP performed the analyses described in the manuscript. All authors helped to write the manuscript.

References

1. Shipp MA, Ross KN, Tamayo P, Weng AP, Kutok JL, Aguiar RCT, et al. Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. Nature Medicine. 2002;8:68–74.

2. Nutt CL, Mani DR, Betensky RA, Tamayo P, Cairncross JG, Ladd C, et al. Gene Expression-based Classification of Malignant Gliomas Correlates Better with Survival than Histological Classification. Cancer Res. 2003;63:1602–7.

3. Yuan Y, Van Allen EM, Omberg L, Wagle N, Amin-Mansour A, Sokolov A, et al. Assessing the clinical utility of cancer genomic and proteomic data across tumor types. Nature Biotechnology. 2014;32:644–52.

4. Bilal E, Dutkowski J, Guinney J, Jang IS, Logsdon BA, Pandey G, et al. Improving Breast Cancer Survival Analysis through Competition-Based Multidimensional Modeling. PLoS Computational Biology. 2013;9:e1003047.

5. Piccolo SR, Andrulis IL, Cohen AL, Conner T, Moos PJ, Spira AE, et al. Gene-expression patterns in peripheral blood classify familial breast cancer susceptibility. BMC Medical Genomics. 2015;8:72.

6. Piccolo SR, Frey LJ. Clinical and molecular models of glioblastoma multiforme survival. International Journal of Data Mining and Bioinformatics. 2013;7:245–65.

7. Desautels T, Calvert J, Hoffman J, Jay M, Kerem Y, Shieh L, et al. Prediction of Sepsis in the Intensive Care Unit With Minimal Electronic Health Record Data: A Machine Learning Approach. JMIR Medical Informatics. 2016;4:e28.

8. Szlosek DA, Ferrett J. Using Machine Learning and Natural Language Processing Algorithms to Automate the Evaluation of Clinical Decision Support in Electronic Medical Record Systems. EGEMS (Washington, DC). 2016;4:1222.

9. Statnikov A, Aliferis CF, Tsamardinos I, Hardin D, Levy S. A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. Bioinformatics. 2005;21:631–43.

10. Kim J-W, Sharma V, Ryan ND. Predicting Methylphenidate Response in ADHD Using Machine Learning Approaches. The International Journal of Neuropsychopharmacology. 2015;18:pyv052.

11. Braman NM, Etesami M, Prasanna P, Dubchuk C, Gilmore H, Tiwari P, et al. Intratumoral and peritumoral radiomics for the pretreatment prediction of pathological complete response to neoadjuvant chemotherapy based on breast DCE-MRI. Breast Cancer Research: BCR. 2017;19:57.

12. Libbrecht MW, Noble WS. Machine learning applications in genetics and genomics. Nature Reviews Genetics. 2015;16:321–32.

13. Pedregosa F, Varoquaux G. Scikit-learn: Machine learning in Python. 2011.

14. Frank E, Hall M, Holmes G, Kirkby R, Pfahringer B, Witten IH, et al. Weka-A Machine Learning Workbench for Data Mining. In: Maimon O, Rokach L, editors. Data Mining and Knowledge Discovery Handbook. 2nd ed. New York: Springer Science+Business Media, LLC; 2010. pp. 1269–77.

15. Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Jones Z, et al. Mlr: Machine Learning in R. 2016.

16. Piccolo SR, Frey LJ. ML-Flex : A Flexible Toolbox for Performing Classification Analyses In Parallel. Journal of Machine Learning Research. 2012;13:555–9.

17. Kuhn M, others. Building predictive models in R using the caret package. Journal of Statistical Software. 2008;28:1–26.

18. Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, et al. KNIME: The Konstanz Information Miner. Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007). Springer; 2007.

19. Guyon I, Elisseeff A. An Introduction to Variable and Feature Selection. Journal of Machine Learning Research. 2003;3:1157–82.

20. Dougherty ER, Hua J, Sima C. Performance of Feature Selection Methods. Current Genomics. 2009;10:365–74.

21. Varma S, Simon R. Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics. 2006;7:91.

22. Statnikov A, Wang L, Aliferis CF. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. BMC Bioinformatics. 2008;9:319.

23. Fernández-Delgado M, Cernadas E, Barro S, Amorim D. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? Journal of Machine Learning Research. 2014;15:3133–81.

24. Piccolo SR, Frampton MB. Tools and techniques for computational reproducibility. GigaScience. 2016;5:30.

25. Wickham H. Tidy Data. Journal of Statistical Software. 2014;59.

26. Docker. Docker. https://www.docker.com/. Accessed June 2019.

27. Chollet F, others. Keras. https://keras.io/. Accessed 18 June 2019.

28. Cook D. Practical machine learning with H2O: Powerful, scalable techniques for deep learning and AI. " O'Reilly Media, Inc."; 2016.

29. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. TensorFlow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016. pp. 265–83.

30. Python Software Foundation. Python Language Reference, version 3.6. 2013. https://docs.python.org/release/3.6.0/ Accessed 18 June 2019.

31. R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing; 2019. https://www.r-project.org/ Accessed 18 June 2019.

32. Java Software | Oracle. https://www.oracle.com/java/. Accessed 18 June 2019.

33. Fowler M. Continuous Integration. martinfowler.com. https://martinfowler.com/articles/continuousIntegration.html. Accessed 18 June 2019.

34. Chang W, Cheng J, Allaire JJ, Xie Y, McPherson J. Shiny: Web Application Framework for R. 2019. https://cran.r-project.org/web/packages/shiny/index.html Accessed 18 June 2019.

35. Harris D, Harris S. Digital design and computer architecture. Morgan Kaufmann; 2010.

36. Geisser S. The Predictive Sample Reuse Method with Applications. Journal of the American Statistical Association. 1975;70:320–8.

37. Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. International Joint Conference on Artificial Intelligence. 1995;14:1137–45.

38. Green DM, Swets JA, others. Signal detection theory and psychophysics. Wiley New York; 1966.

39. Brier GW. Verification of forecasts expressed in terms of probability. Monthly Weather Review. 1950;78:1–3.

40. Vickery B. Reviews: Van Rijsbergen, CJ Information retrieval. 2nd edn. London, Butterworths, I978. 208pp. Journal of librarianship. Sage Publications Sage CA: Thousand Oaks, CA; 1979;11:237–7.

41. Matthews BW. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA) - Protein Structure. 1975;405:442–51.

42. Ballings M, Van den Poel D. AUC: Threshold independent performance measures for probabilistic classifiers. 2013. https://cran.r-project.org/web/packages/AUC/index.html. Accessed 15 June 2018.

43. Wickham H, François R, Henry L, Müller K. Dplyr: A Grammar of Data Manipulation. 2018.

44. Dowle M, Srinivasan A. Data.Table: Extension of ‘data.Frame‘. 2018.

45. Wickham H, Hester J, Francois R. Readr: Read Rectangular Text Data. 2018.

46. Black D. Partial justification of the Borda count. Public Choice. 1976;28:1–15.

47. Wickham H. Ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York; 2016.

48. Wilke CO. Cowplot: Streamlined Plot Theme and Plot Annotations for ’ggplot2’. 2017.

49. Lecun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521:436–44.

50. Rosenblatt F. Principles of neurodynamics. Perceptrons and the theory of brain mechanisms. Cornell Aeronautical Lab Inc Buffalo NY; 1961.

51. Chen T, Guestrin C. Xgboost: A scalable tree boosting system. Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM; 2016. pp. 785–94.

52. Quinlan JR. Induction of decision trees. Machine Learning. 1986;1:81–106.

53. Fan R-E, Chang K-W, Hsieh C-J, Wang X-R, Lin C-J. LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research. 2008;9:1871–4.

54. Vapnik VN. Statistical learning theory. New York: Wiley; 1998.

55. Hulten G, Spencer L, Domingos P. Mining time-changing data streams. ACM SIGKDD Intl Conf On Knowledge Discovery and Data Mining. ACM Press; 2001. pp. 97–106.

56. Landwehr N, Hall M, Frank E. Logistic Model Trees. 2005;95:161–205.

57. Kruskal WH, Wallis WA. Use of ranks in one-criterion variance analysis. Journal of the American statistical Association. Taylor & Francis Group; 1952;47:583–621.

58. Ishwaran H, Kogalur UB, Kogalur MUB. Package “randomForestSRC”. 2019

59. Cover TM, Thomas JA. Elements of information theory. John Wiley & Sons; 2012.

60. Breiman L. Random Forests. Machine Learning. 2001;45:5–32.

61. Guyon I, Weston J, Barnhill S, Vapnik V. Gene selection for cancer classification using support vector machines. Machine Learning. 2002;46:389–422.

62. Pearson K. VII. Note on regression and inheritance in the case of two parents. proceedings of the royal society of London. The Royal Society London; 1895;58:240–2.

63. Holte RC. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. Machine Learning. 1993;11:63–90.

64. Kononenko I. Estimating attributes: Analysis and extensions of RELIEF. In: Bergadano F, De Raedt L, editors. Machine Learning ECML94. Berlin / Heidelberg: Springer; 1994. pp. 171–82.

65. Witten IH, Frank E, Hall MA, Pal CJ. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann; 2016.

66. Walt S van der, Colbert SC, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering. 2011;13:22–30.

67. Olson RS, La Cava W, Orzechowski P, Urbanowicz RJ, Moore JH. PMLB: A large benchmark suite for machine learning evaluation and comparison. BioData Mining. 2017;10:36.

68. Simonoff JS. Analyzing Categorical Data. New York: Springer-Verlag; 2003.

69. Michalski RS, Mozetic I, Hong J, Lavrac N. The Multi-purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains. Proceedings of the Fifth AAAI National Conference on Artificial Intelligence. AAAI Press; 1986. pp. 1041–5.

70. Güvenir HA, Demiröz G, Ilter N. Learning differential diagnosis of erythemato-squamous diseases using voting feature intervals. Artificial Intelligence in Medicine. 1998;13:147–65.

71. Diaconis P, Efron B. Computer-intensive methods in statistics. Scientific American. Scientific American, a division of Nature America, Inc. 1983;248:116–31.

72. Fisher RA. The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics. 1936;7:179–88.

73. Robinson D, Allaway SL, Ritchie CD, Smolski OR, Bailey AR. The use of artificial intelligence in the prediction of alcohol-induced fatty liver. MEDINFO 89: Proceedings of the Sixth Conference on Medical Informatics, Beijing, China, 16-20 October 1989 and Singapore, Republic of Singapore, 11-15 December 1989. North Holland; 1989. p. 170.

74. Harley CB, Reynolds RP. Analysis of E. Coli promoter sequences. Nucleic Acids Research. 1987;15:2343–61.

75. Horton P, Nakai K. A probabilistic classification system for predicting the cellular localization sites of proteins. Proceedings International Conference on Intelligent Systems for Molecular Biology. 1996;4:109–15.

76. Sklearn.Preprocessing.RobustScaler scikit-learn 0.21.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html; Accessed 18 June 2018.

77. Ho TKT, Hull J, Srihari SNS, Member S. Decision combination in multiple classifier systems. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1994;16:66–75.

78. Demsar J, Zupan B, Leban G, Curk T. Orange: From experimental machine learning to interactive data mining. Knowledge Discovery in Databases: PKDD 2004. Berlin; 2004. pp. 537–9.

79. Elgohary A, Boehm M, Haas PJ, Reiss FR, Reinwald B. Compressed linear algebra for large-scale machine learning. Proceedings of the VLDB Endowment. VLDB Endowment; 2016;9:960–71.

80. Boettiger C. An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review. 2015;49:71–9.

81. Ho YY, Pepyne D. Simple Explanation of the No-Free-Lunch Theorem and Its Implications. Journal of Optimization Theory and Applications. 2002;115:549–70.

82. Xu L, Krzyzak A, Suen C. Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Transactions on Systems Man and Cybernetics. 1992;22:418–35.

83. Stephen Piccolo (2019) ShinyLearner Demo [Source Code]. https://doi.org/10.24433/CO.5449763.v1

84. Stephen Piccolo, Kimball Hill, Erica Suh, & Jonathan Dayton. (2019, November 15). srp33/ShinyLearner: Gigascience (Version 1). Zenodo. http://doi.org/10.5281/zenodo.3543724

85. Stephen Piccolo. (2019, November 15). srp33/ShinyLearner_gpu: Gigascience (Version 1). Zenodo. http://doi.org/10.5281/zenodo.3543726

86. Stephen Piccolo. (2019, November 15). srp33/ShinyLearner_Environment_gpu: Gigascience (Version 1). Zenodo. http://doi.org/10.5281/zenodo.3543728

87. Stephen Piccolo. (2019, November 15). srp33/ShinyLearner_Environment: Gigascience (Version 1). Zenodo. http://doi.org/10.5281/zenodo.3543730

88. Piccolo SR; Lee TJ; Suh E; Hill K (2020): Supporting data for "ShinyLearner: A containerized benchmarking tool for machine-learning classification of tabular data" GigaScience Database. http://dx.doi.org/10.5524/100701

89. Eglen, Stephen J. (2020, February 18). CODECHECK Certificate 2020-001. Zenodo. http://doi.org/10.5281/zenodo.3674056

Figure 1

```
mkdir -p "/home/user/OutputData"

docker run --rm -i \
  -v "/home/user/InputData":"/InputData" \
  -v "/home/user/OutputData":"/OutputData" \
  --user $(id -u):$(id -g) \
  srp33/shinylearner:version511 \
  /UserScripts/nestedclassification_montecarlo \
    --data "data.tsv" \
    --description "My_Analysis_Description" \
    --outer-iterations 10 \
    --inner-iterations 5 \
    --classif-algo "/AlgorithmScripts/Classification/tsv/keras/dnn/*" \
    --classif-algo "/AlgorithmScripts/Classification/tsv/mlr/h2o.randomForest/*" \
    --classif-algo "/AlgorithmScripts/Classification/tsv/mlr/xgboost/*" \
    --classif-algo "/AlgorithmScripts/Classification/tsv/sklearn/svm/*" \
    --seed 1 \
    --ohe true \
    --scale true \
    --impute false
```

Figure 4

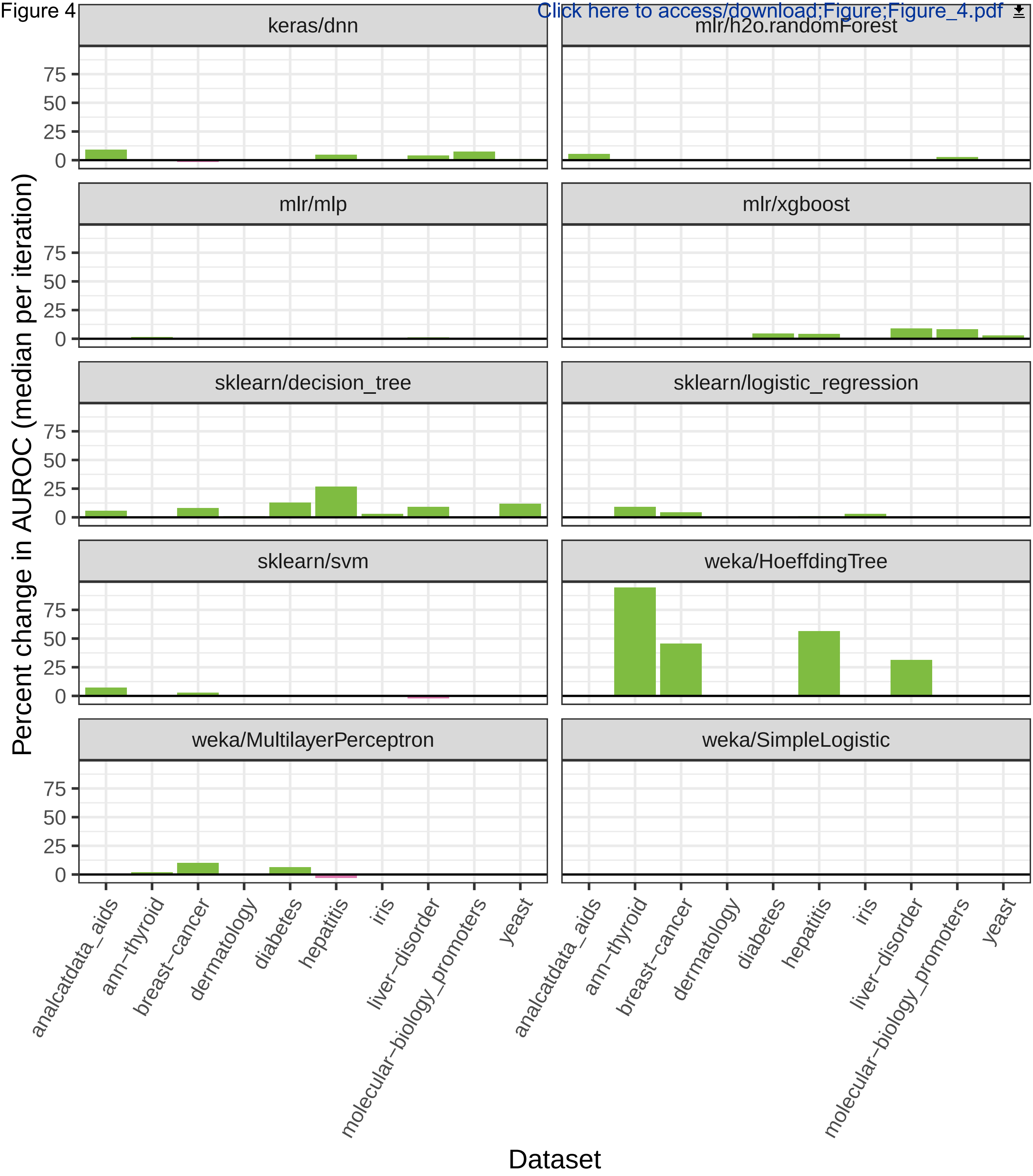

keras/dnn
mlr/h2o.randomForest
mlr/mlp
mlr/xgboost
sklearn/decision_tree
sklearn/logistic_regression
sklearn/svm
weka/HoeffdingTree
weka/MultilayerPerceptron
weka/SimpleLogistic
75
50
25
0
Percent change in AUROC (median per iteration)
analcatdata_aids
ann−thyroid
breast−cancer
dermatology
diabetes
hepatitis
iris
liver−disorder
molecular−biology_promoters
yeast
Dataset

Figure 3

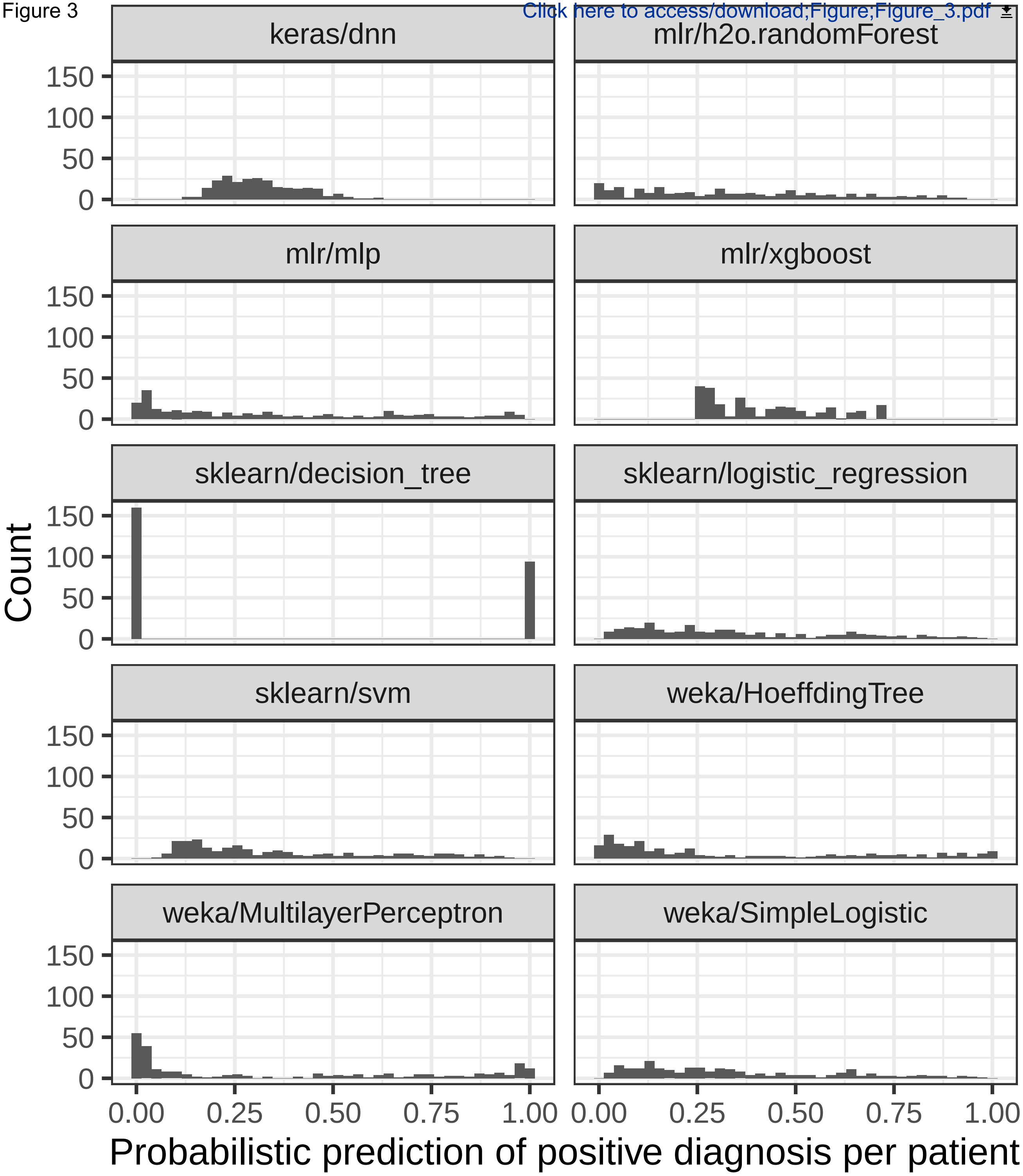
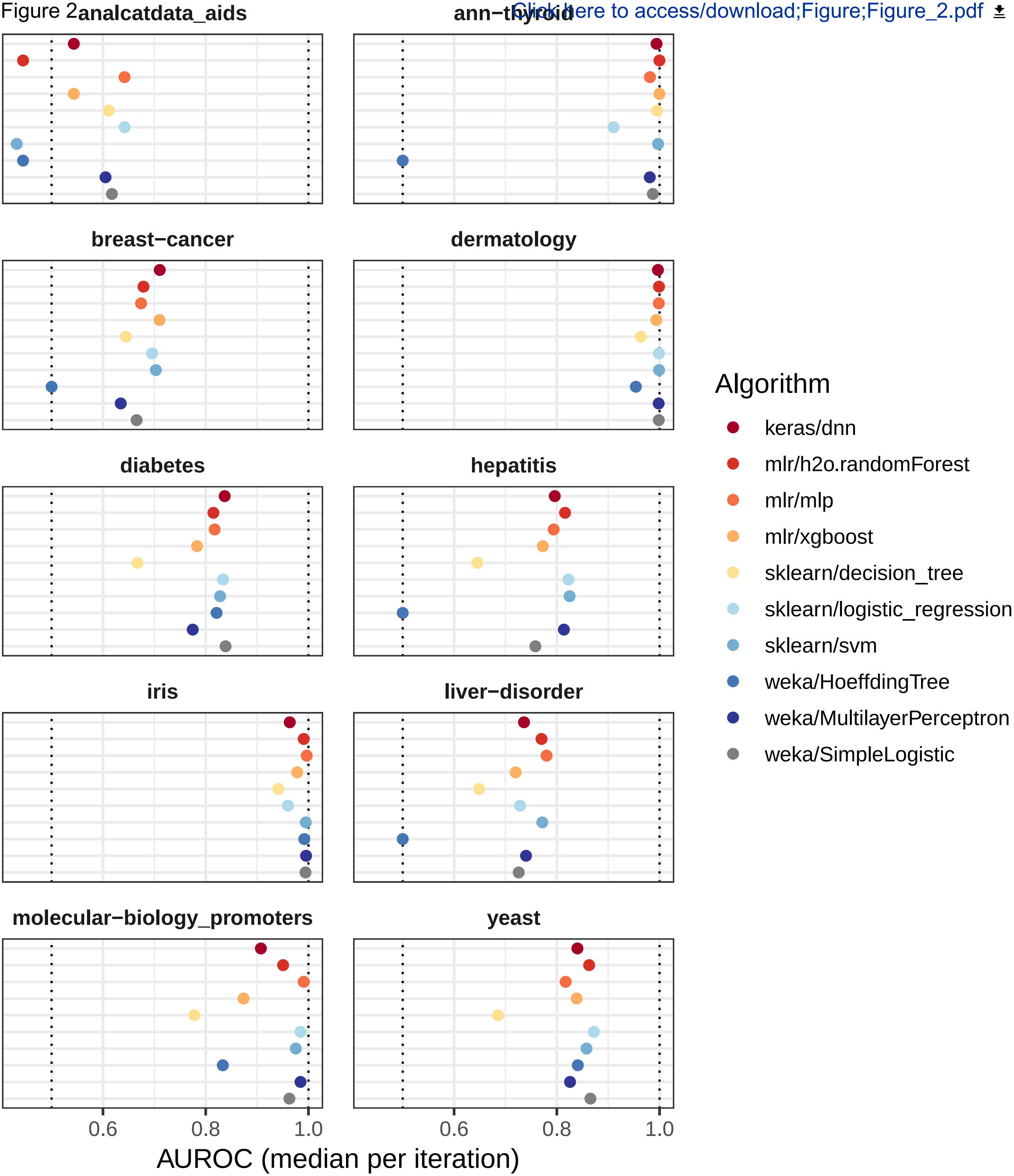

keras/dnn
mlr/h2o.randomForest
mlr/mlp
mlr/xgboost
sklearn/decision_tree
sklearn/logistic_regression
sklearn/svm
weka/HoeffdingTree
weka/MultilayerPerceptron
weka/SimpleLogistic
Count
150
100
50
0
0.00
0.25
0.50
0.75
1.00
Probabilistic prediction of positive diagnosis per patient

Figure 2



analcatdata_aids
ann−thyroid
breast−cancer
dermatology
diabetes
hepatitis
iris
liver−disorder
molecular−biology_promoters
yeast
0.6
0.8
1.0
0.6
0.8
1.0
AUROC (median per iteration)
Algorithm
keras/dnn
mlr/h2o.randomForest
mlr/mlp
mlr/xgboost
sklearn/decision_tree
sklearn/logistic_regression
sklearn/svm
weka/HoeffdingTree
weka/MultilayerPerceptron
weka/SimpleLogistic

Figure 5

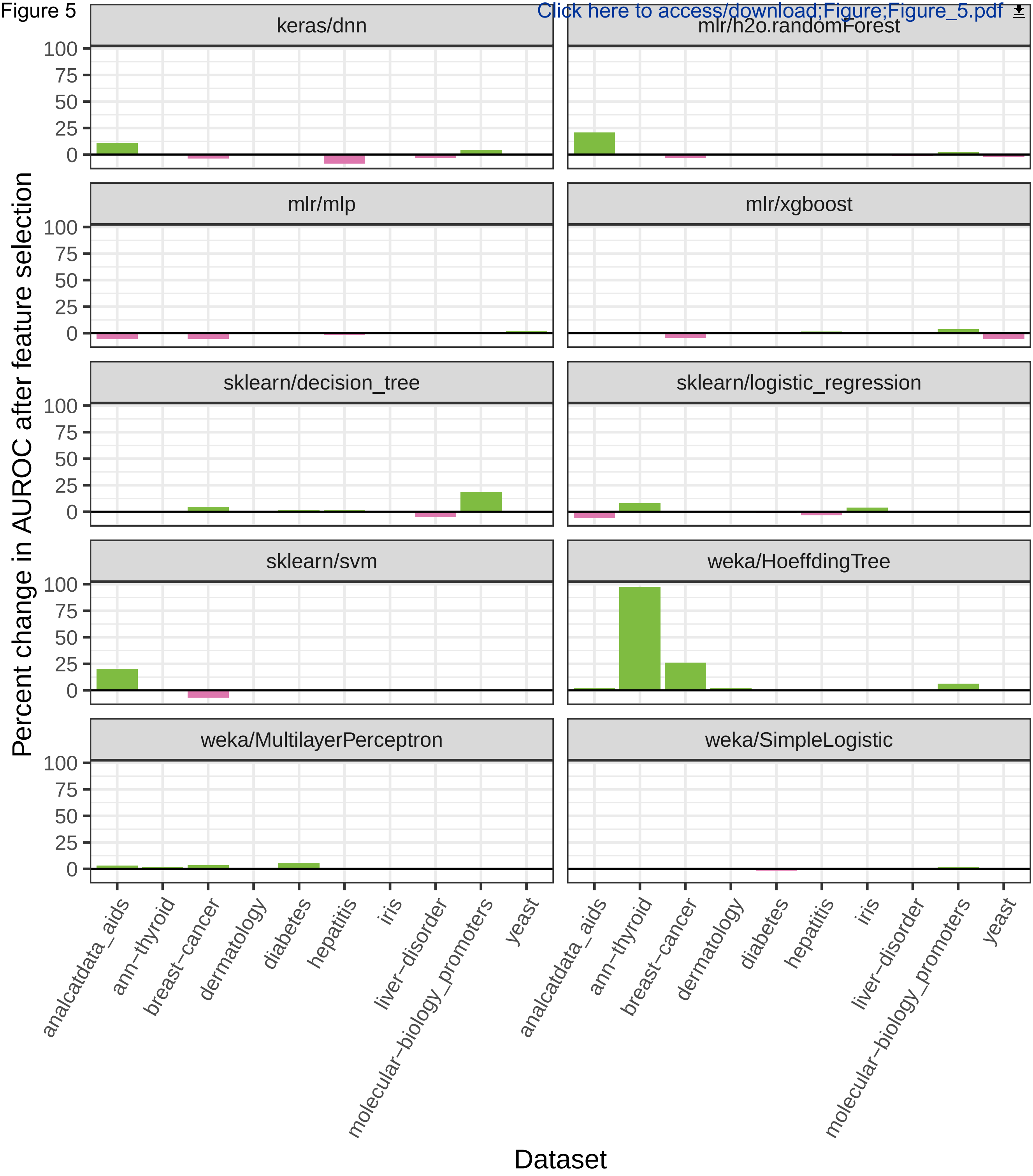

keras/dnn
mlr/h2o.randomForest
mlr/mlp
mlr/xgboost
sklearn/decision_tree
sklearn/logistic_regression
sklearn/svm
weka/HoeffdingTree
weka/MultilayerPerceptron
weka/SimpleLogistic
Percent change in AUROC after feature selection
100
75
50
25
0
analcatdata_aids
ann−thyroid
breast−cancer
dermatology
diabetes
hepatitis
iris
liver−disorder
molecular−biology_promoters
yeast
Dataset

Figure 6

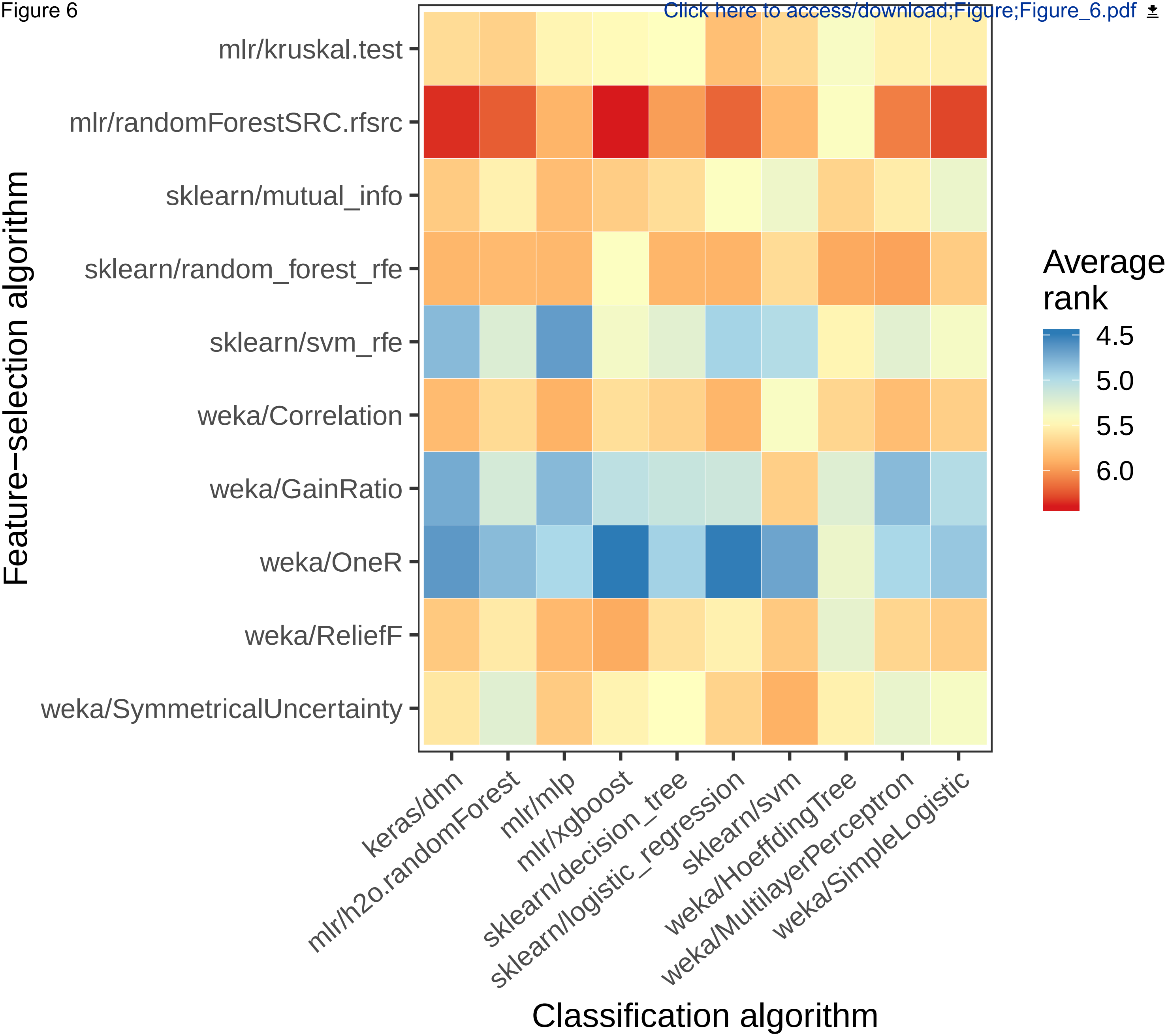
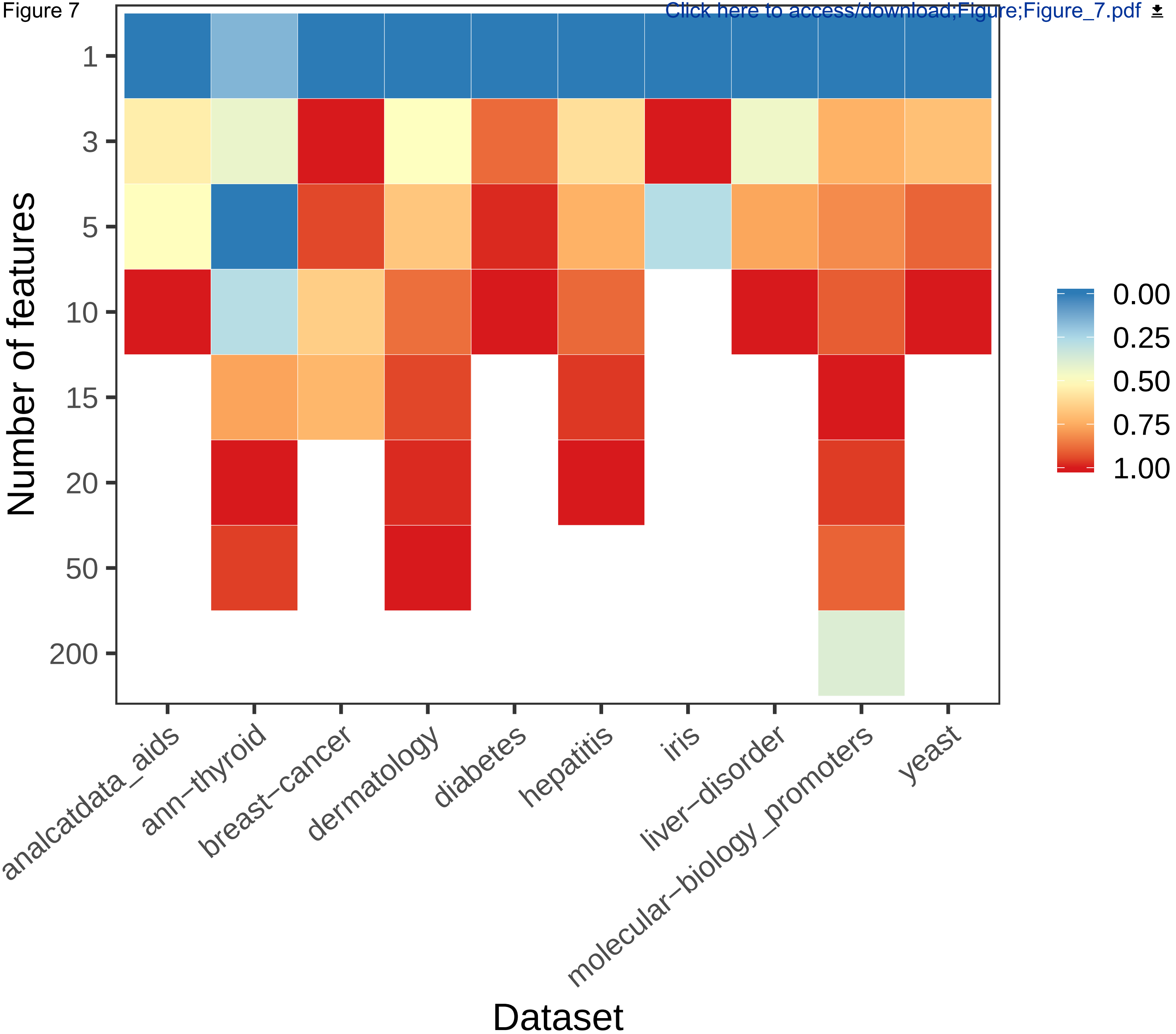

Feature−selection algorithm
mlr/kruskal.test
mlr/randomForestSRC.rfsrc
sklearn/mutual_info
sklearn/random_forest_rfe
sklearn/svm_rfe
weka/Correlation
weka/GainRatio
weka/OneR
weka/ReliefF
weka/SymmetricalUncertainty
keras/dnn
mlr/h2o.randomForest
mlr/mlp
mlr/xgboost
sklearn/decision_tree
sklearn/logistic_regression
sklearn/svm
weka/HoeffdingTree
weka/MultilayerPerceptron
weka/SimpleLogistic
Classification algorithm
Average rank
4.5
5.0
5.5
6.0

Figure 7


Number of features
1
3
5
10
15
20
50
200
Dataset
analcatdata_aids
ann−thyroid
breast−cancer
dermatology
diabetes
hepatitis
iris
liver−disorder
molecular−biology_promoters
yeast
0.00
0.25
0.50
0.75
1.00

DEPARTMENT OF BIOLOGY

Dear Scott Edmunds, Ph.D.:

Thank you for overseeing the review of our manuscript entitled, "*ShinyLearner: A containerized benchmarking tool for machine-learning classification*." We suggest this manuscript as a **Technical Note**, but we are flexible on the format.

We have registered ShinyLearner at bio.tools (https://bio.tools/ShinyLearner; ID: biotools:ShinyLearner) and at SciCrunch.org (RRID: SCR_017608). We mention this in the manuscript.

We have also provided a point-by-point response to the reviewers. The revised manuscript conforms to the journal style.

Warm regards,

Stephen R. Piccolo, PhD
Assistant Professor
Department of Biology
Brigham Young University
(801) 422-7116
Stephen_Piccolo@byu.edu

Reviewer #1: The manuscript by Piccolo et al presents a new tool for comparing classification algorithms in a consistent interface. A key strength of the current approach is that it uses Docker containers so that once a user has installed Docker, a wide variety of algorithms from different machine learning toolboxes can be compared. The typical user for this package would be someone new to machine learning wishing to compare a wide range of algorithms on their data.

Thank you for taking the time to write this review and for providing this positive feedback!

I am grateful to the authors for providing online resources; in particular the Code-Ocean capsule worked for me (after first logging in; presumably anonymous access is not possible), and all the figures from the manuscript were regenerated from the analysis results. (But if the online Rmd is to be updated, the output figures could be made taller as some of the y-axes were too squashed to see the range of data, e.g. I could not see the negative, red, differences for Figure 5 online).

Yes, as far as we understand it, anonymous access is not possible via Code Ocean. Thank you for the feedback about some of the figures being squashed. This is true for the HTML output file, produced from the Rmd file. However, we don't know of a way to control those dimensions. If you look at the right-hand panel of the output in Code Ocean, you should be able to see PDF files that are generated from the R code. The names of those files coincide with names specified in the Rmd file, so you should be able to match those and see what they look like in the correct dimensions.

What I couldn't do however, was to re-run the analysis. https://github.com/srp33/ShinyLearner/blob/master/Demo/Execute_Algorithms.ipynb lists the script, and it looks appropriate, but as I'm already late with this review, I'll have to assume it works. (How long does it take to execute?) Could it also be hosted on CODE OCEAN, or would it take too long?

It takes several days for this to execute. Although Code Ocean does not put hard limits on execution times, it's not really designed for such long-running executions.

The text from https://github.com/srp33/ShinyLearner/blob/master/Word_of_Caution.md is important and should be copied into the discussion section of the paper.

Thank you for this suggestion. We have added a slightly modified version of this text to the end of the Discussion section in our manuscript.

Although all the code is available on github, I think an archive of the github repo should be stored on Zenodo to give a permanent DOI of the repository when (assuming) the manuscript is published.

We have created Zenodo archives of the relevant GitHub repositories for this tool and have added the DOI for these archives to the manuscript under "Availability of source code and requirements".

The online tool http://bioapps.byu.edu/shinylearner/ looks great, but again due to being late with this review, I didn't get time to run it yet for myself. It would of course help for the very first time that there is a demo where I could download some data first (e.g. the Iris dataset, or MNIST) to work through this. Am I right in assuming that the role of the GUI is to build the eventual docker command to then be run locally?

You are correct that the role of the GUI is to build the eventual Docker command, which would be run locally. The GUI provides a link to a page on our GitHub site (https://github.com/srp33/ShinyLearner/blob/master/InputFormats.md), which describes input formats that can be used and provides example data files.

Minor:

I found it confusing to constantly flip between the main and supplementary figures. If a figure is important, please could it be folded into the main document? The first figure reference is on line 104, and that is to S1 (showing the docker command line invocation), rather than Figure 1 of the paper.

In the main part of the manuscript, we included what we thought would be figures that are most interesting to readers and relegated the remaining figures to the supplementary material. It just happened that the first figure we referenced was a supplementary figure, but the order in which the figures are mentioned does not necessarily indicate the figure's importance (as with any paper). Having said that, we can see the value in including the first supplementary figure in the main body, and we have made that change. When submitting our updated manuscript, we have uploaded the figures as separate files.

line 42: SUPPORT --> REQUIRE

This change has been made.

Is figure 2 required? It was obvious from figure 1 (to me at least) that the HoeffdingTree and decision_tree algorithms were lagging behind the others.

This figure has been moved to the Supplementary Material. We feel that it will be useful to some readers, even though it overlaps somewhat with the other figure.

lines 281-284: You show here that there are a few differences between algorithms that should be working the same. Did you explore why there were small differences? Parameter settings or initialisation methods? (I'm not surprised there are small differences, but thought you could explain them.)

We have added the following text for clarification: "For both of these algorithms, the available hyperparameters as well as options that users can select are considerably different between the underlying machine-learning libraries."

Figure 7: what classifier was used to do this analysis?

We have revised the caption for this figure to indicate that all 10 classification algorithms were used and that we averaged the results across these.

In the discussion, (line 389-396), six reasons supporting use of ShinyLearner are presented. I am convinced of the first two reasons, but I think most competent programmers would feel that they could also investigate points 3--6 in their own environment. Unless of course you are arguing that only ShinyLearner provides the wide diversity of algorithms that is absent in one environment (like R or Python). However, if you are to make this case, I think you need to point out specific examples of e.g. what classes of methods (rather than implementations) are missing e.g. in R or Python. My hunch, but happy to be proven wrong, is that R and Python each provide pretty much close to a full toolkit of machine learning methods.

To address these questions, we have added the following statement to the manuscript: "Although many of these tasks could be performed by a researcher who has programming expertise, care must be taken to ensure that the steps are performed in a robust manner (e.g., not mixing training and test sets in nested validation). In addition, we hope ShinyLearner will increase the efficiency of such benchmark studies by reducing duplicate efforts."

Figure S1: it looks like you are punching holes from Docker into the user's directory. I think you need to explain any potential security risks here.

This should not be a security concern. To share data between the host operating system and the container, the user executing the container must have permission to read and write data in the host directories. Using commands such as those recommended by our GUI will ensure that the container can only read/write files for which the user has permission. There are other potential security concerns with Docker, but we feel that a discussion of those topics is outside the scope of this manuscript.

Figure S2: explain vertical dotted lines in legend.

We have added this to the caption.

Figure S3 (and S4): Are the Coefficients of Variation simply (s.d. / mean) or have they been multiplied by 100 to be a percentage?

The Coefficient of Variation is often expressed as a percentage in academic papers. We have modified the figure captions to clarify that these are percentages.

Figure S8: what does color denote?

We have clarified this in the caption.

Figure S10: Took me a while to work out the three coloured curves are for the three patients; perhaps rework last sentence of legend to make this clearer.

We have clarified this in the caption.

The word "Shiny" in the title should be explained somewhere to refer to the Shiny R package for making GUIs.

We have modified the text to explain this.

Stephen Eglen

Reviewer #2: The authors present a very compelling tool that allows researchers to compare and benchmark algorithms across various packages regardless of their design. Shinylearner also provides a simple web interface that allows users to easily generate Docker commands that anyone can execute (assuming they have a basic knowledge of Docker).

I found the rationale of the paper to be very clear and coherent. The authors make a clear case for the need for benchmarking, especially when comparing disparate algorithms and their various software implementations, especially from the lens of applying supervised learning to biomedical studies.

ShinyLearner is modular and extensive, and language agnostic, making it widely useful to a broad community of researchers. It also allows for diverse inputs and standardizes outputs. By running all benchmarks inside a container, it removes the challenges of dealing with complex dependencies, especially for users that may not be technically savvy, all the while not building out a black box. It supports a large number of classification algorithms, has gpu support and is already set up to be sustainable.

We thank the reviewer for these positive and encouraging comments!